Extending a HSF-enabled Open-Source RTOS with Resource Sharing
– International Workshop OSPERT 2010 –

M.M.H.P. van den Heuvel - R.J. Bril - J.J. Lukkien - M. Behnam

System Architecture and Networking (SAN)
Department of Mathematics and Computer Science
Eindhoven University of Technology
The Netherlands

6 July 2010
Outline

1. Introduction

2. Task Synchronization

3. Inter-subsystem resource sharing

4. Conclusions
Many embedded devices provide **multiple, integrated functionalities**.

In such systems it's important to deliver correct functionality **on time**.

- **Non-real-time systems**
  - Correct function if produced result is correct

- **Real-time systems**
  - Correct function if produced result is correct and **delivered on time**

These functionalities share both **logical** and **physical resources**.
Integration Problem

1. Isolation: applications shall not *interfere*
   - Temporal isolation (processor);
   - Spatial isolation (memory).
1. Isolation: applications shall not *interfere*
   - Temporal isolation (processor);
   - Spatial isolation (memory).

2. Development and analysis versus integration
   - Independent analysis of application on *virtual platforms*;
   - Application specific scheduling algorithms;
   - Composition of virtual platforms.
Integration Problem

1. Isolation: applications shall not *interfere*
   - Temporal isolation (processor);
   - Spatial isolation (memory).

2. Development and analysis versus integration
   - Independent analysis of application on *virtual platforms*;
   - Application specific scheduling algorithms;
   - Composition of virtual platforms.

3. Applications may share logical resources.
A Solution: Hierarchical Scheduling

- subsystem: server, set of tasks and local (task) scheduler
- server: a budget allocated each period
A Solution: Hierarchical Scheduling

- subsystem: server, set of tasks and local (task) scheduler
- server: a budget allocated each period

Tasks, located in arbitrary subsystems, may share logical resources
Outline

1 Introduction

2 Task Synchronization

3 Inter-subsystem resource sharing

4 Conclusions
MicroC/OS-II Basics

MicroC/OS-II is

- a commercial RTOS
- targeted at embedded systems
- open source
- available at http://micrium.com/

It provides

- a portable and configurable kernel
- a fixed-priority, preemptive task scheduler
- basic services (mailboxes, mutexes and counting semaphores)
Visualization of scheduling behavior:


- MicroC/OS-II port to OpenRISC platform

- OpenRISC: Architectural Simulator
  http://opencores.org/openrisc, or1ksim
microC/OS-II’s mutexes: Priority calling

Priority calling is similar to priority inheritance protocol (PIP):

**Priority Inheritance Rule:**
when a higher-priority task blocks on a resource, the lower-priority task holding the resource inherits the higher priority;
Priority calling is similar to *priority inheritance protocol* (PIP):

**Priority Inheritance Rule:**
when a higher-priority task blocks on a resource, the lower-priority task holding the resource inherits the higher priority;

<table>
<thead>
<tr>
<th>Priority Calling</th>
<th>Priority Inheritance</th>
</tr>
</thead>
<tbody>
<tr>
<td>unique priority for each resource</td>
<td>run-time priority adaption</td>
</tr>
<tr>
<td>inherits a predefined priority</td>
<td>transitive priority inheritance</td>
</tr>
<tr>
<td>non-transparent</td>
<td>transparent</td>
</tr>
</tbody>
</table>
Priority calling is similar to *priority inheritance protocol* (PIP):

**Priority Inheritance Rule:**
when a higher-priority task blocks on a resource, the lower-priority task holding the resource inherits the higher priority;

<table>
<thead>
<tr>
<th>Priority Calling</th>
<th>Priority Inheritance</th>
</tr>
</thead>
<tbody>
<tr>
<td>unique priority for each resource</td>
<td>run-time priority adaption</td>
</tr>
<tr>
<td>inherits a predefined priority</td>
<td>transitive priority inheritance</td>
</tr>
<tr>
<td>non-transparent</td>
<td>transparent</td>
</tr>
</tbody>
</table>

**It is not:** Highest Locker Protocol (HLP) or Stack Resource Policy (SRP).

- **microC/OS-II:** a task inherits a higher priority *only* when a higher priority task is blocked;
- **in HLP/SRP** a task immediately inherits a priority *when it locks a resource.*
microC/OS-II's synchronization protocol suffers from deadlock:

Legend:
- active
- holding
- mutex

Conclusions:
- microC/OS-II implements a non-transparent, priority-inheritance protocol.
microC/OS-II: protocol classification

microC/OS-II’s synchronization protocol suffers from deadlock:

Conclusion:
microC/OS-II implements a non-transparent, priority-inheritance protocol
Intermezzo: Stack Resource Policy (SRP)

- Each resource has a statically determined resource ceiling:

**Definition of a resource ceiling:**
the maximum priority of any task that could use the resource.
Each resource has a statically determined resource ceiling:

**Definition of a resource ceiling:**
the maximum priority of any task that could use the resource.

A dynamically updated system ceiling is maintained:

**Definition of the system ceiling**
the maximum resource ceiling of any resource currently being locked in the system.
Intermezzo: Stack Resource Policy (SRP)

- Each resource has a statically determined *resource ceiling*:

**Definition of a resource ceiling:**
the maximum priority of any task that could use the resource.

- A dynamically updated *system ceiling* is maintained:

**Definition of the system ceiling**
the maximum resource ceiling of any resource currently being locked in the system.

- A task can only be selected for execution if
  1. it has the highest priority among all ready tasks;
  2. its priority is higher than the current system ceiling.
SRP provides non-blocking primitives:
- therefore it allows tasks to share their execution stack;
- blocking occurs upon an attempt to preempt, rather than upon an attempt to access a resource.
SRP provides non-blocking primitives:
- therefore it allows tasks to share their execution stack;
- blocking occurs upon an attempt to preempt, rather than upon an attempt to access a resource.

SRP is non-transparent, similar as microC/OS-II’s PIP-like implementation.
SRP provides non-blocking primitives:
- therefore it allows tasks to share their execution stack;
- blocking occurs upon an attempt to preempt, rather than upon an attempt to access a resource.

SRP is non-transparent, similar as microC/OS-II’s PIP-like implementation.

Maintaining the system ceiling can be implemented using a stack data structure:
- we stack the resource ceilings of used resources in a monotonically increasing order;
- the top of the stack represents the system ceiling.
microC/OS-II: Our SRP extension

Deadlocks are resolved:

Legend:
- active
- holding
- mutex

Easy implementations (approx. 170 lines of code);
Extended microC/OS-II scheduler with SRP's preemption rule.
microC/OS-II: Our SRP extension

Deadlocks are resolved:

Easy implementations (approx. 170 lines of code);
Extended microC/OS-II scheduler with SRP’s preemption rule.
Outline

1 Introduction

2 Task Synchronization

3 Inter-subsystem resource sharing

4 Conclusions
Global resource sharing problem

Budget depletion during a critical section can lead to excessive blocking times:

- SRP locally
- SRP globally

Task1
Task2
Task3
OS-ServerIdle

Legend:
- active
- holding mutex

Server1
Server2
Global resource sharing problem

Two SRP-based solutions for fixed-priority scheduling:

- **HSRP**: React upon budget depletion while a resource is locked; i.e. allow to use an overrun budget
  1. **with payback**: the consumed overrun budget is subtracted from the next budget provisioning;
  2. **no payback**: no penalty for overrun consumption.
Global resource sharing problem

Two SRP-based solutions for fixed-priority scheduling:

- **HSRP:** React upon budget depletion while a resource is locked; i.e. allow to use an overrun budget
  1. **with payback:** the consumed overrun budget is subtracted from the next budget provisioning;
  2. **no payback:** no penalty for overrun consumption.

- **SIRAP:** Prevent budget depletion during resource access; i.e. before granting access, first check the remaining budget.
HSRP provides overrun budget (optionally a payback mechanism):

Legend:
- active
- holding mutex

Task1
Task2
Server-1-Idle
Task3
Task4
Server-2-Idle
OS-ServerIdle

Server1
Server2

Legend:
SIRAP uses a skipping mechanism:

Legend:
- active
- holding mutex
## HSRP and SIRAP implementation overhead and issues

<table>
<thead>
<tr>
<th>Event</th>
<th>HSRP</th>
<th>SIRAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock resource</td>
<td>-</td>
<td>spinlock</td>
</tr>
<tr>
<td>Unlock resource</td>
<td>overrun completion</td>
<td>-</td>
</tr>
<tr>
<td>Budget depletion</td>
<td>overrun</td>
<td>-</td>
</tr>
<tr>
<td>Budget replenishment</td>
<td>overrun completion, payback (optionally)</td>
<td>spinlock-completion</td>
</tr>
</tbody>
</table>

**HSRP**: close to default SRP; expensive queue manipulations to track overrun budget; complex implementation due to explicit event handling.

**SIRAP**: spinlocking is executed within a task's context, but wastes budget; alternatively: suspend (i.e. block) and resume a task, but this is not SRP-compliant!
HSRP and SIRAP implementation overhead and issues

<table>
<thead>
<tr>
<th>Event</th>
<th>HSRP</th>
<th>SIRAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock resource</td>
<td>-</td>
<td>spinlock</td>
</tr>
<tr>
<td>Unlock resource</td>
<td>overrun completion</td>
<td>-</td>
</tr>
<tr>
<td>Budget depletion</td>
<td>overrun</td>
<td>-</td>
</tr>
<tr>
<td>Budget replenishment</td>
<td>overrun completion, payback (optionally)</td>
<td>spinlock-completion</td>
</tr>
</tbody>
</table>

- **HSRP:**
  - close to default SRP;
  - expensive queue manipulations to track overrun budget;
  - complex implementation due to explicit event handling.
HSRP and SIRAP implementation overhead and issues

<table>
<thead>
<tr>
<th>Event</th>
<th>HSRP</th>
<th>SIRAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock resource</td>
<td>-</td>
<td>spinlock</td>
</tr>
<tr>
<td>Unlock resource</td>
<td>overrun completion</td>
<td>-</td>
</tr>
<tr>
<td>Budget depletion</td>
<td>overrun</td>
<td>-</td>
</tr>
<tr>
<td>Budget replenishment</td>
<td>overrun completion, payback (optionally)</td>
<td>spinlock-completion</td>
</tr>
</tbody>
</table>

- **HSRP:**
  - close to default SRP;
  - expensive queue manipulations to track overrun budget;
  - complex implementation due to explicit event handling.

- **SIRAP:**
  - spinlocking is executed within a task’s context, but wastes budget;
  - alternatively: suspend (i.e. block) and resume a task,
  - but this is not SRP-compliant!
Conclusions

We presented:

- a classification of microC/OS-II’s synchronization protocol;
- an efficient task-level SRP implementation;
- two alternative hierarchical SRP-implementations, i.e. SIRAP and HSRP;
- a side-by-side integration of SIRAP and HSRP in a single HSF.

We made a minimal number of modifications to MicroC/OS-II.

Upcoming work:
- EDF-based synchronization (including BROE);
- protocol-transparent global resource sharing.
Conclusions

We presented:

- a classification of microC/OS-II’s synchronization protocol;

- an efficient task-level SRP implementation;

- two alternative hierarchical SRP-implementations, i.e. SIRAP and HSRP;

- a side-by-side integration of SIRAP and HSRP in a single HSF.

We made a minimal number of modifications to MicroC/OS-II.
Conclusions

We presented:

- a classification of microC/OS-II’s synchronization protocol;
- an efficient task-level SRP implementation;
- two alternative hierarchical SRP-implementations, i.e. SIRAP and HSRP;
- a side-by-side integration of SIRAP and HSRP in a single HSF.

We made a minimal number of modifications to MicroC/OS-II.

Upcoming work:

- EDF-based synchronization (including BROE);
- protocol-transparent global resource sharing.

